Post-mortem memory leak diagnosis with ELF and Virtual Functions

Peter Edwards, Arista
peadar@arista.com

ARISTA

Scene:

- Large, long-running process, serving HTTP requests.
- C-with-classes
- No memory diagnostics designed in to speak of
- Customer running a few thousand transactions per second through process
- Process slowly leaking memory
- Cumulative effect meant process would vomit up a core after a day or so.
- Not reproducible in test

Where’s the hole in the bucket?

- Could potentially provide an instrumented build, but:
  - conservative customers won’t always agree to this easily
  - may reduce performance below acceptable threshold
- Core file is the best thing we have to work with
- There’s always ‘strings’
  - `$ cat core | strings | sort | uniq -c | sort -n`

C++ vtables

For classes with virtual functions:

- There's one virtual function table
- Each instance of that type starts with a pointer to the table
- The compiler will generate a symbol for the table, eg:
  - "vtable for Namespace::Type"
- Mangled for the linker:
  - _ZTVN9Namespace4TypeE

C++ vtables - concept

```
class Base {
public:
   virtual void f();
   virtual void g() = 0;
   virtual void h();
   virtual ~Base();
};

class Derived : public Base {
public:
   virtual void f();
   virtual void g();
   virtual ~Derived();
};
```

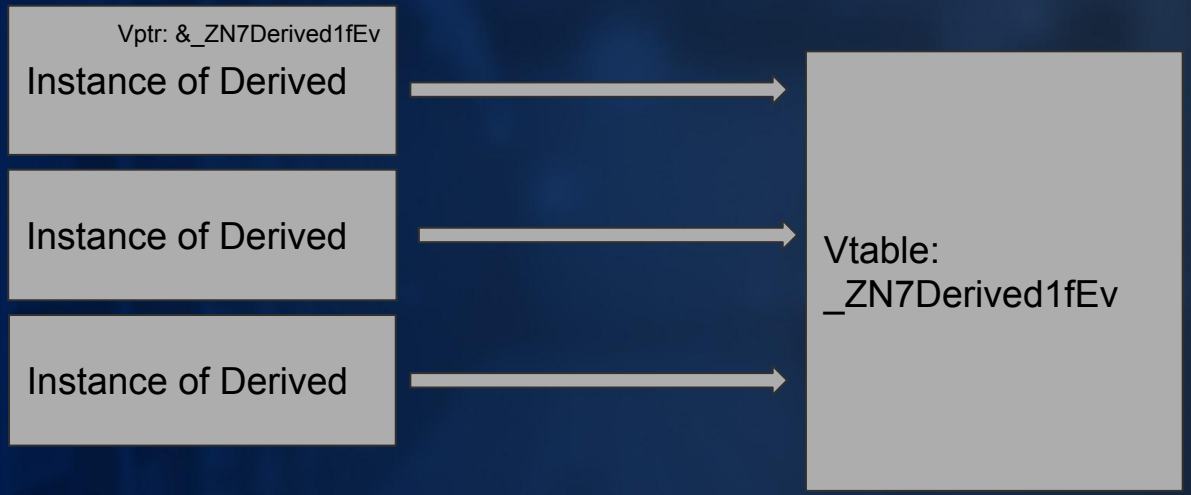
C++ vtables - generated vtable

```
_ZTV7Derived:
        .quad   0
        .quad   _ZTI7Derived
        .quad   _ZN7Derived1fEv
        .quad   _ZN7Derived1gEv
        .quad   _ZN4Base1hEv
        .quad   _ZN7DerivedD1Ev
        .quad   _ZN7DerivedD0Ev
```

C++ vtables - generated table, thru c++filt

```
vtable for Derived:
        .quad   0
        .quad   typeinfo for Derived
        .quad   Derived::f()
        .quad   Derived::g()
        .quad   Base::h()
        .quad   Derived::~Derived()
        .quad   Derived::~Derived()
```

vptrs and vtables



Setting vptrs

- Calling a virtual function when constructing or destructing a base type calls the base type's virtual functions, as for destruction
- Implemented by overwriting the vptr to the more derived type after the base is constructed, and vice versa on destruct
- Each instance of a live object has a pointer to its vtable
- These 32- or 64- bit values are unlikely to occur otherwise.

Core files

$objdump -x ./corefile

….

Program Header:
NOTE off 0x0000000000000740 vaddr 0x0000000000000000 paddr 0x0000000000000000 align 2**0
filesz 0x0000000000000e64 memsz 0x0000000000000000 flags ---
LOAD off 0x0000000000002000 vaddr 0x0000000000400000 paddr 0x0000000000000000 align 2**12
filesz 0x0000000000001000 memsz 0x0000000000003000 flags r-x
LOAD off 0x0000000000003000 vaddr 0x0000000000602000 paddr 0x0000000000000000 align 2**12
filesz 0x0000000000001000 memsz 0x0000000000001000 flags r--
LOAD off 0x0000000000004000 vaddr 0x0000000000603000 paddr 0x0000000000000000 align 2**12
filesz 0x0000000000001000 memsz 0x0000000000001000 flags rw-
LOAD off 0x0000000000005000 vaddr 0x0000000002465000 paddr 0x0000000000000000 align 2**12
filesz 0x0000000000032000 memsz 0x0000000000032000 flags rw-
LOAD off 0x0000000000037000 vaddr 0x00007fa04b603000 paddr 0x0000000000000000 align 2**12
filesz 0x0000000000001000 memsz 0x00000000001c0000 flags r-x

Finding our memory leak

- Any leaked object on the heap that has virtual functions will have a telltale pointer at the start of the allocated memory to the vtbl for that type
- Process:
  - Find all the symbols in the image that look like vtables, i.e.,
  - Walk the entire core, 4 bytes at a time, looking for pointers to the vtables

Demo

ARISTA

Questions

https://github.com/peadar/pstack

https://github.com/peadar/pstack/blob/master/canal.cc